Crear Servicios para Windows con Python

INTRODUCCIÓN

En este artículo vamos a ver como crear *servicios de windows* con el lenguaje de programación *Python*.

Lo primero es entender qué es un *servicio*.

¿qué es un servicio en informática?

Existen dos timpos de procesos que puede ejecutar el ordenador.

1. Aplicaciones
2. Servicios / Daemons

Aplicaciones

Las *aplicaciones* son aquellos procesos que se ejecutan generalmente en primer plano y responden a la interacción con el usuario. Son *los programas* que todo el mundo conoce.

Pueden presentar interfaz gráfica o no. Si no tienen interfaz gráfica se ejecutan en el *terminal* (*cmd*, *command prompt*, etc).

O sea, una *aplicación* está vinculada a un *terminal*. Este *terminal* puede ocultarse, lo que generalmente es deseable si la *aplicación* tiene interfaz gráfica.

El hecho de que una *aplicación* esté vinculada a un *terminal* significa que una aplicación **solamente podrá ejecutarse una vez que el usuario ha iniciado sesión**. Esto, a su vez, significa que una *aplicación* **tiene los permisos del usuario que la está ejecutando**.

Si el usuario es *administrador*, su aplicación podrá hacer cualquier cosa e incluso puede *ocultar su terminal*, siendo el programa *invisible* si carece de interfaz gráfica. Pero la *aplicación*, **por muchos permisos que tenga y por muy invisible que sea no es un servicio, ya que se ejecuta dentro de la sesión de un usuario y solamente puede operar a nivel de usuario**.

Servicios / Daemons

Los *servicios* (llamados *daemons* en *Linux*), son un **tipo de procesos que se ejecutan de forma ajena al usuario**. Los *servicios* no están vinculados al usuario, por lo que **se pueden ejecutar antes de que se haya iniciado sesión**.

Por otra parte, al ser *independientes del usuario*, los servicios *no pretenden interactuar con el usuario*, sino realizar una función en *segundo plano*. **No tienen interfaz gráfica y no están vinculados a un *terminal***.

Un *servicio* puede programarse para **hacer cualquier cosa**, ya que no depende de un usuario.

CREAR UN SERVICIO CON PYTHON

Ahora que entendemos cual es la diferencia entre una *aplicación* y un *servicio*, la buena noticia es que **puedes crear windows-services con python**. El proceso para crear un *servicio para windows con python* es el siguiente:

1. Crea un *script* que realize una determinada tarea.

   En principio debes crear una *aplicación de consola*, sin interfaz gráfica. El *script* debe funcionar **sin la intervención del usuario**. Posteriormente instalaremos un *servicio* que ejecutará este *script*, es decir, nuestro *script* finalmente se estará ejecutando como un *servicio* y no como una *aplicación de consola*.

2. Instalar el *servicio* que ejecutará la tarea.

   La forma mas fácil de *instalar servicios en windows* es mediante *NSSM - the Non-Sucking Service Manager*. Esta **utilidad gratuita** permite instalar, iniciar, detener y desinstalar *servicios de windows*. Con *nssm* instalaremos un *windows-service* que ejecutará nuestro *script* al iniciarse *Windows* (pero antes de que el usuario inicie sesión, pues el *servicio* es un *proceso* ajeno al usuario y sus permisos).

Empecemos creando un *script* que realice una determinada tarea.

UN PROGRAMA QUE BLOQUEA PROCESOS

Vamos a crear un *script python* capaz de bloquear los procesos que no deseemos que se ejecuten en nuestro PC. Crea un *script* llamado *proclocker.py* y copia el siguiente código:

```
import psutil

for proc in psutil.process_iter():
    print(proc.name(), proc.pid)
```

Si no tienes instalado el *módulo psutil* puedes instalarlo mediante:

```
python -m pip install psutil
```

Al ejecutar esta sencilla *aplicación* se imprimirán en consola todos los procesos activos, cada uno de ellos seguido de su *id*. Para nuestra *aplicación* (que finalmente será un *servicio*) solamente nos interesa el nombre de los procesos activos: `proc.name()`

Ahora vamos a interceptar un proceso. Por ejemplo, si queremos sabes si *chrome.exe* está ejecutándose podemos hacer lo siguiente:

```
import psutil

for proc in psutil.process_iter():
    if proc.name() == 'chrome.exe':
        print('Chrome Detectado')
```

Para evitar errores debidos a que el nombre de un proceso puede tener tanto caracteres en mayúsculas como en minúsculas podemos mejorar la comprobación anterior utilizando el método `str.lower`:

```
import psutil

for proc in psutil.process_iter():
    if proc.name().lower() == 'chrome.exe':
        print('Chrome Detectado')
```

Ahora vamos a hacer que la *aplicación* reciba el nombre del proceso que queremos interceptar mediante *argumentos de terminal*:

```
import sys
import psutil

if len(sys.argv) == 1:
    print('Este programa no funciona sin argumentos')
    sys.exit(0)

target = sys.argv[1]

for proc in psutil.process_iter():
    if proc.name().lower() == target:
        print(f'{target} Detectado')
```

Ahora podemos ejecutar el programa indicando el proceso que queremos interceptar por línea de comandos:

```
python proclocker.py chrome.exe
```

Para evitar errores al intentar obtener `sys.argv[1]` en caso de ejecutar el programa sin argumentos he añadidos la siguiente comprobación al inicio del *script*:

```
if len(sys.argv) == 1:
    print('Este programa no funciona sin argumentos')
    sys.exit(0)
```

Con este condicional conseguimos evitar el error, de manera que si no se indica el *target* como argumento la *aplicación* nos avisa y se cierra.

Ahora vamos a hacer que *el proceso indicado por línea de comandos se cierre si es localizado*. Mediante `proc.kill()` podemos detener un proceso:

```
for proc in psutil.process_iter():
    if proc.name().lower() == target.lower():
        proc.kill()
        print(f'{target} Detenido')
```

Si tienes *Chrome* abierto y ahora ejecutas:

```
python proclocker.py chrome.exe
```

¿que sucede? *Google Chrome* se ha cerrado. Perfecto! Puedes probar a cerrar *firefox.exe* o cualquier otro proceso.

Observa que ahora la comprobación realizada es `proc.name().lower() == target.lower()`, de modo que da igual si el argumento pasado al programa es *chrome.exe*, *CHROME.EXE*, *Chrome.exe*, *Chrome.EXE*, etc.

Para organizar el código vamos a definir una *función lock* que englobe la funcionalidad del *bucle que itera sobre los procesos activos*:

```
import sys
import psutil

def check_arguments():
    if len(sys.argv) == 1:
        print('Este programa no funciona sin argumentos')
        sys.exit(0)

def lock(target):
    for proc in psutil.process_iter():
        if proc.name().lower() == target.lower():
                proc.kill()


if __name__ == '__main__':

    check_arguments()
    target = sys.argv[1]
    lock(target)
```

Además, ahora la comprobación de argumentos se realiza en la función `check_arguments`. Al ejecutarse el *script* directamente (cuando no está siendo importado como módulo) se cumple la condición `__name__ == '__main__'`, de modo que se comprueban los argumentos. Si existen argumentos se asigna `target = sys.argv[1]` y se ejecuta la *función lock* pasándole *target* como argumento: `lock(target)`.

Como lo que queremos crear es un *servicio de windows*, nos interesa que el programa cierre los procesos que indiquemos siempre que se abran (no solamente al ejecutar el *script*), por lo que podemos aplicar la siguiente modificación:

```
if __name__ == '__main__':

    check_arguments()
    target = sys.argv[1]

    while True:
        lock(target)
```

Si ahora ejecutas:

```
python proclocker.py chrome.exe
```

¿que sucede? si *chrome* está abierto se cierra inmediatamente, pero además ahora si intentas abrir *chrome* este se cierra inmediatamente. `lock(target)` se está ejecutando en un *bucle infinito*.

Vamos a aplicar una última modificación al *script* antes de *instalarlo como servicio de windows*. Vamos a hacer que acepte varios parámetros para poder especificar varios procesos que deseamos cerrar en caso de que estén abiertos:

```
if __name__ == '__main__':

    check_arguments()
    targets = sys.argv[1:]

    while True:
        for target in targets:
                lock(target)
```

Ahora puedes ejecutar el programa para que monitoré varios procesos y que los cierre si intentan abrirse. Por ejemplo:

```
python proclocker chrome.exe firefox.exe riotclientux.exe
```

Al ejecutar el programa con estos argumentos se impide que el usuario abra *Chrome*, *Firefox* y *League of Legends*.

Puedes modificar el programa para que acepte argumentos sin tener que escribir constantemente *.exe*:

```
def get_targets():

    targets = sys.argv[1:]

    i = 0
    while i < len(targets):

        if not targets[i].endswith('.exe'):
                targets[i] = targets[i] + '.exe'

        i += 1

    return targets
```

Ahora puedes obtener los *targets* mediante `targets = get_targets()`, de manera que sería posible ejecutar el programa, por ejemplo, así:

```
python proclocker.py chrome firefox.exe riotclientux.exe
```

Es decir, puedes omitir *.exe* en los argumentos.

El programa proclocker.py queda finalmente así:

```
import sys
import psutil

def check_arguments():
    if len(sys.argv) == 1:
        print('Este programa no funciona sin argumentos')
        sys.exit(0)

def get_targets():
```

```
    targets = sys.argv[1:]

    i = 0
    while i < len(targets):

        if not targets[i].endswith('.exe'):
                targets[i] = targets[i] + '.exe'

        i += 1

    return targets

def lock(target):
    for proc in psutil.process_iter():
        if proc.name().lower() == target.lower():
                proc.kill()


if __name__ == '__main__':

    check_arguments()
    targets = get_targets()

    while True:
        for target in targets:
                lock(target)
```

Nuestra *aplicación* está lista, pero si la cierras deja de funcionar. La *consola de comandos* es visible y es muy fácil que alguien evite *el bloqueo*.

Ha llegado la hora de *instalar proclocker.py como un servicio*.

INSTALAR EL PROGRAMA COMO UN SERVICIO

Voy a suponer que el *path* del archivo *proclocker.py* es **C:\custom-services\proclocker.py**.

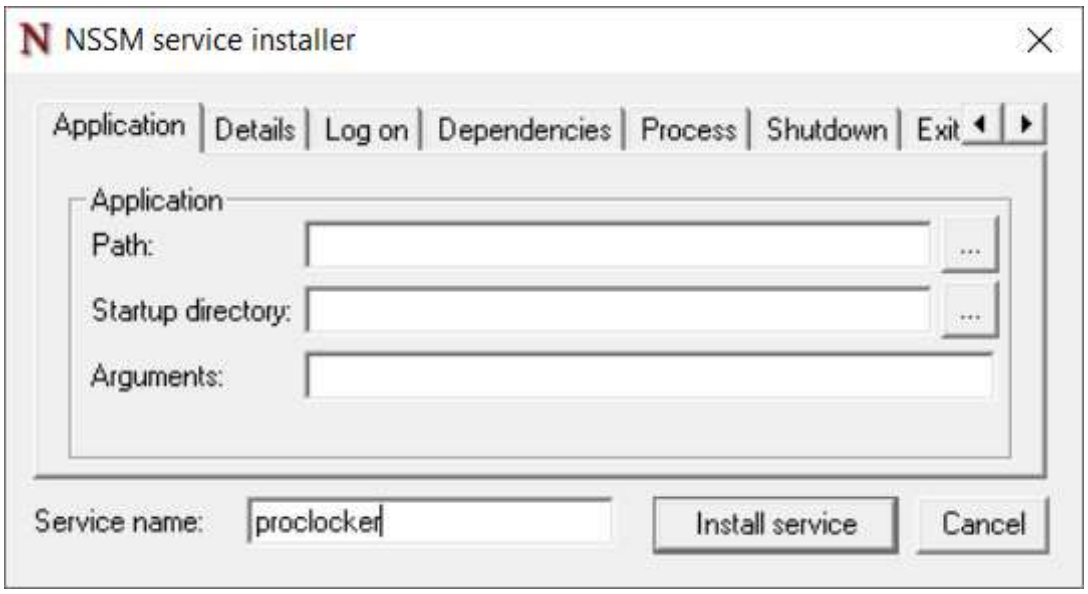
1. *Descarga Non-Sucking Service Manager*.
2. Descomprime el *.zip* que acabas de descargar.
3. Copia **nssm.exe** en **C:\custom-services**:
    - Si tu PC es de 32bits, copia el *nssm.exe* de la carpeta **win32.**
    - Si tu PC es de 64bits, copia el *nssm.exe* de la carpeta **win64.**
4. Abre el *terminal de comandos* como administrador y navega a la carpeta de trabajo:

```
cd C:\custom-services
```

5. Abre el *instalador de nssm* ejecutando:

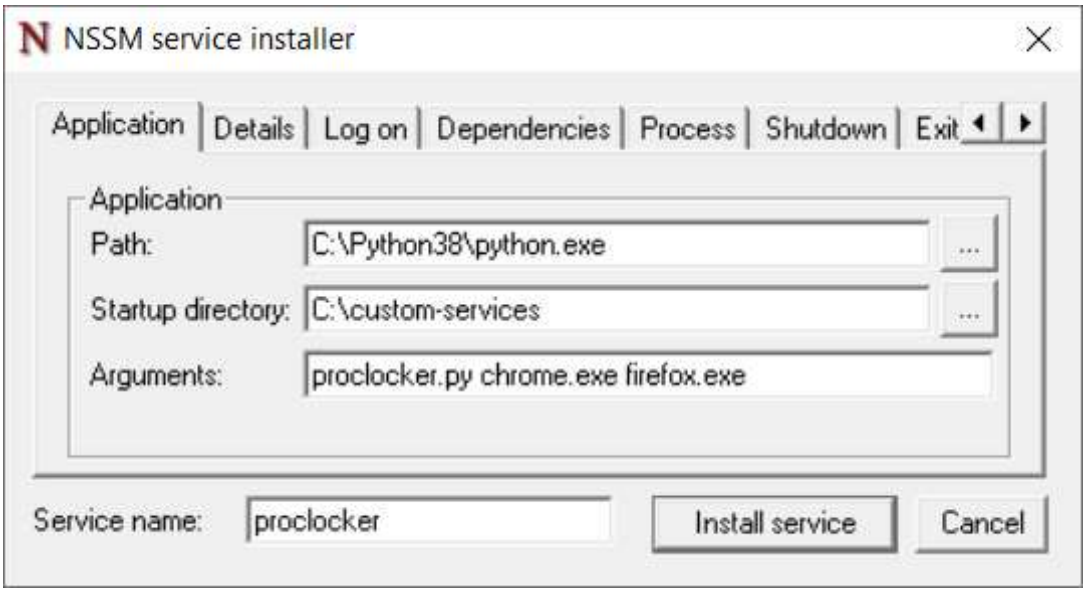
```
nssm.exe install proclocker
```

Se abrirá el siguiente formulario:

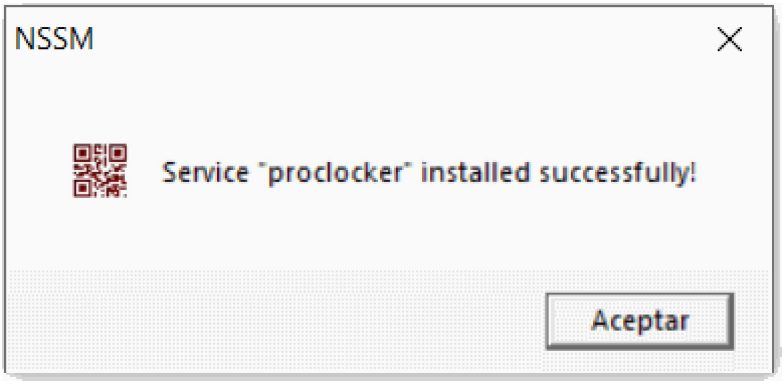


Rellena los campos *Path*, *Startup directory* y *Arguments*:

- **Path:** Indica el *path del intérprete de python*. En mi caso: **C:\Python38\python.exe**
- **Startup directory:** Indica el directorio en el que se encuentra *proclocker.py*: **C:\custom-services**
- **Arguments:** Por ejemplo, si quieres que el servicio intercepte a *Chrome* y *Firefox*: **proclocker.py chrome.exe firefox.exe**. Como en el campo *Startup directory* hemos indicado el directorio en el que se encuentra el *script proclocker.py* no es necesario pasar el *path absoluto del script* como primer argumento, y en su lugar he indicado el *path relativo*: **proclocker.py**.



Finalmente, haz click en el botón **[Install service]** para instalar el servicio. Se abrirá el siguiente *cuadro de diálogo*:



Y se imprimirá el siguiente mensaje en la *consola*:

```
Service "proclocker" installer successfully!
```

Una vez instalado el servicio debes iniciarlo desde la *consola*:

```
nssm.exe start proclocker
```

Se imprimirá el siguiente texto en la *consola* indicando que el *servicio* se ha iniciado correctamente:

```
proclocker: START: La operación se completó correctamente.
```

Ahora **no puedes abrir *Chrome* ni *Firefox***. Al intentarlo, estos procesos **se detienen inmediatamente**.

El *servicio* se iniciará por defecto al *reiniciar el sistema*. Puedes modificar este comportamiento.

DETENER EL SERVICIO PROCLOCKER

Para detener el *servicio* abre el *terminal* como administrador y navega a la carpeta donde se encuentra *proclocker.py*:

```
cd C:\custom-services
nssm.exe stop proclocker
```

En la *consola* se imprimirá el siguiente texto que verifica que el *servicio* ha sido detenido correctamente:

```
proclocker: STOP: La operación se completó correctamente.
```

Ahora puedes abrir *Chrome* y *Firefox* sin problemas.

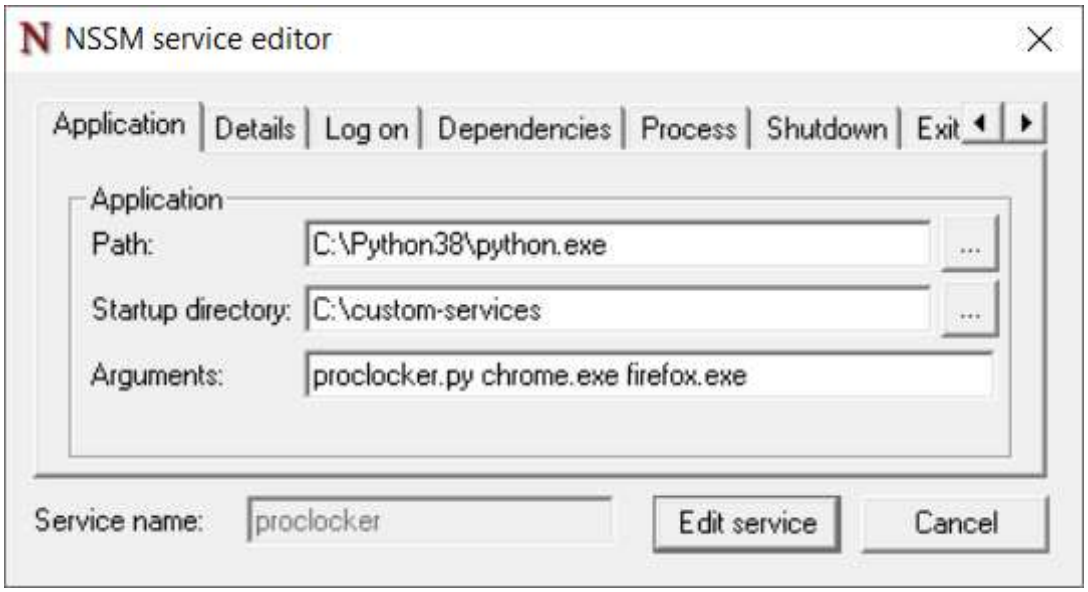
EDITAR EL SERVICIO PROCLOCKER

Antes de editar el *servicio* deberías detenerlo mediante `nssm.exe stop proclocker`.

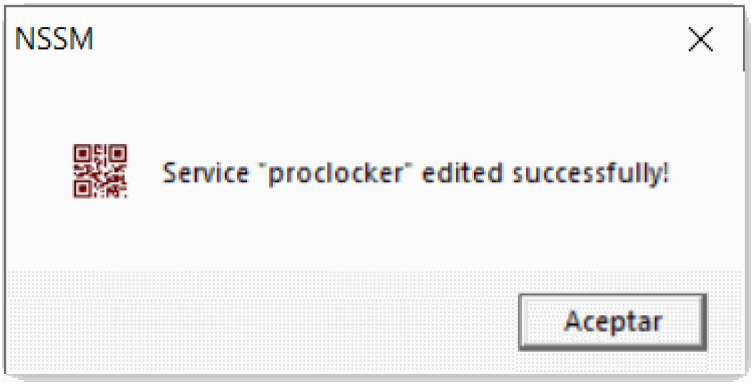
Para editar el servicio abre el *terminal* como administrador y navega a la carpeta donde se encuentra *proclocker.py*:

```
cd C:\custom-services
nssm.exe stop proclocker
nssm.exe edit proclocker
```

Se abrirá el siguiente formulario:



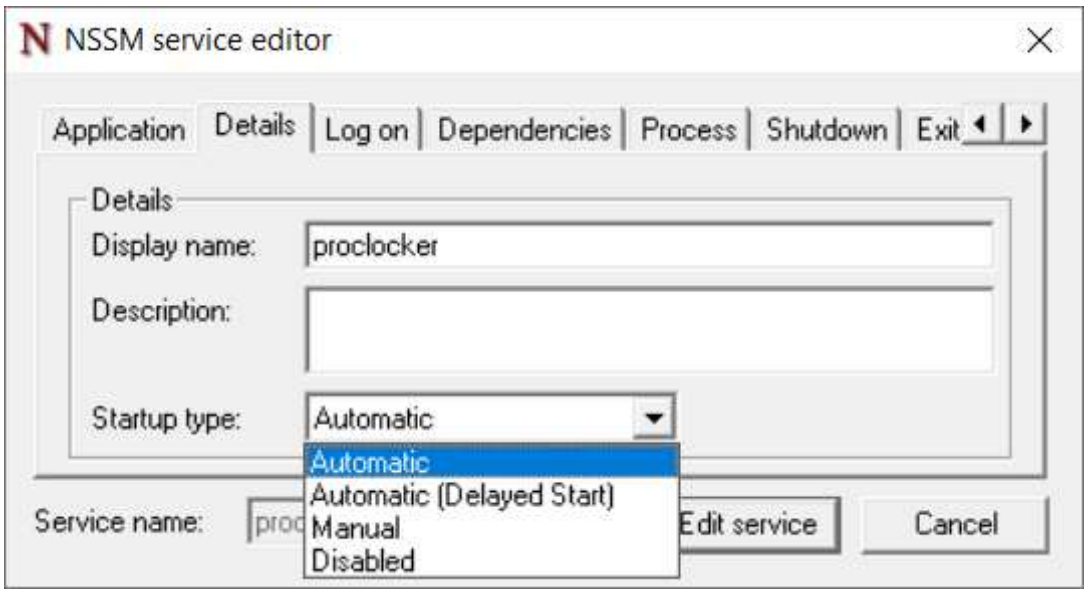
Después de editar el *servicio* simplemente haz click en el botón **[Edit service]** para aplicar los cambios. Se abrirá el sigiente mensaje:



Una vez editado, reinicia el servicio meidante:

```
nssm.exe start proclocker
```

Por defecto, el *servicio* se iniciará automáticamente al *iniciar el sistema*. Puedes modificar este comportamiento desde *NSSM service editor* en la pestaña *Details*:



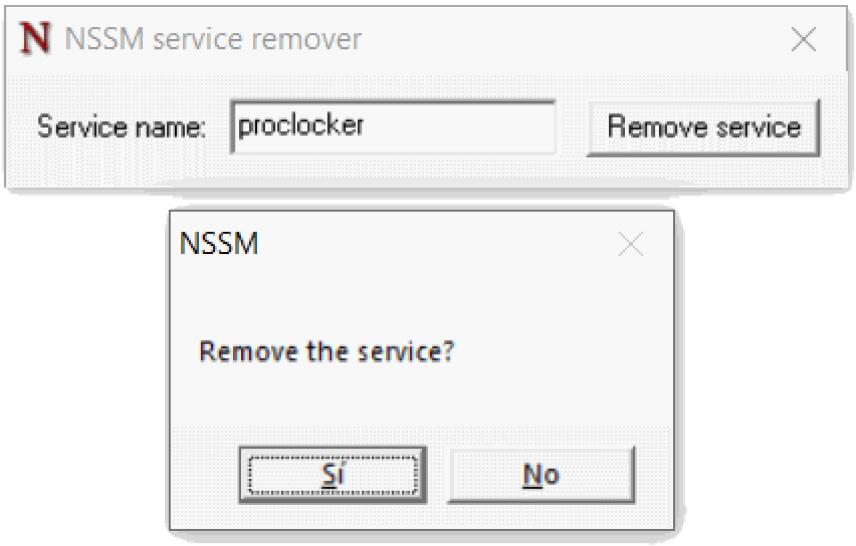
DESINSTALAR EL SERVICIO PROCLOCKER

Antes de desinstalar el *servicio* deberías detenerlo mediante `nssm.exe stop proclocker`.

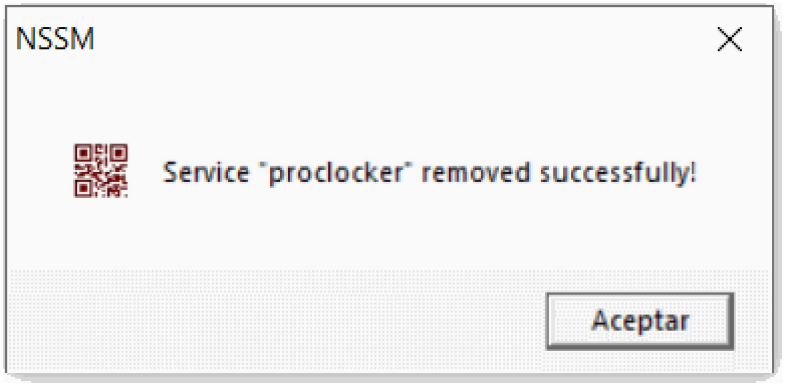
Para desinstalar el servicio abre el *terminal* como administrador y navega a la carpeta donde se encuentra *proclocker.py*:

```
cd C:\custom-services
nssm.exe stop proclocker
nssm.exe remove proclocker
```

nssm pedirá confirmación:



Al confirmar "Sí" aparece el siguiente mensaje:



Y también se notificará por la *consola*:

```
Service "proclocker" removed successfully
```